

Package ‘ClickHouseHTTP’

September 5, 2022

Type Package

Title A Simple HTTP Database Interface to 'ClickHouse'

Version 0.1.3

Description 'ClickHouse' (<<https://clickhouse.com/>>) is an open-source, high performance columnar OLAP (online analytical processing of queries) database management system for real-time analytics using SQL. This 'DBI' backend relies on the 'ClickHouse' HTTP interface and support HTTPS protocol.

URL <https://github.com/patzaw/ClickHouseHTTP>

BugReports <https://github.com/patzaw/ClickHouseHTTP/issues>

Depends R (>= 3.6)

Imports methods, DBI (>= 0.3.0), httr, jsonlite, arrow, data.table

Suggests knitr, rmarkdown, dplyr, stringi

License GPL-3

Encoding UTF-8

RoxygenNote 7.2.1

NeedsCompilation no

Author Patrice Godard [aut, cre, cph]

Maintainer Patrice Godard <patrice.godard@gmail.com>

Repository CRAN

Date/Publication 2022-09-05 07:30:09 UTC

R topics documented:

ClickHouseHTTP	2
ClickHouseHTTPConnection-class	2
ClickHouseHTTPDriver-class	9
ClickHouseHTTPResult-class	11

Index	12
--------------	-----------

ClickHouseHTTP *Create a ClickHouseHTTP DBI driver*

Description

Create a ClickHouseHTTP DBI driver

Usage

ClickHouseHTTP()

Value

A ClickHouseHTTPDriver

See Also

[ClickHouseHTTPDriver](#)

ClickHouseHTTPConnection-class
ClickHouseHTTPConnection class.

Description

ClickHouseHTTPConnection class.

Send SQL query to ClickHouse

Information about the ClickHouse database

Create a table in ClickHouse

Write a table in ClickHouse

Usage

```
## S4 method for signature 'ClickHouseHTTPConnection,character'
dbSendQuery(
  conn,
  statement,
  format = c("Arrow", "TabSeparatedWithNamesAndTypes"),
  file = NA,
  ...
)
```

```
## S4 method for signature 'ClickHouseHTTPConnection'
dbGetInfo(dbObj, ...)
```

```

## S4 method for signature 'ClickHouseHTTPConnection'
dbCreateTable(
  conn,
  name,
  fields,
  engine = "TinyLog",
  overwrite = FALSE,
  ...,
  row.names = NULL,
  temporary = FALSE
)

## S4 method for signature 'ClickHouseHTTPConnection,ANY'
dbWriteTable(
  conn,
  name,
  value,
  overwrite = FALSE,
  append = FALSE,
  engine = "TinyLog",
  ...
)

```

Arguments

conn	a ClickHouseHTTPConnection object created with dbConnect()
statement	the SQL query statement
format	the format used by ClickHouse to send the results. Two formats are supported: "Arrow" (default) and "TabSeparatedWithNamesAndTypes"
file	a path to a file to send along the query (default: NA)
...	Other parameters passed on to methods
dbObj	a ClickHouseHTTPConnection object
name	the name of the table to create
fields	a character vector with the name of the fields and their ClickHouse type (e.g. <code>c("text_col String", "num_col Nullable(Float64)", "nul_col Array(Int32)")</code>)
engine	the ClickHouse table engine as described in ClickHouse documentation . Examples: <ul style="list-style-type: none"> "TinyLog" (default) "MergeTree() ORDER BY (expr)" (expr generally correspond to fields separated by ",")
overwrite	if TRUE and if a table with the same name exists, then it is deleted before creating the new one (default: FALSE)
row.names	unsupported parameter (add for compatibility reason)

- `https`: Is the connection using HTTPS protocol instead of HTTP

`dbCreateTable()` returns TRUE, invisibly.

TRUE; called for side effects

See Also

[ClickHouseHTTPResult](#)

Examples

```
## Not run:

## Connection ----

library(DBI)
### HTTP connection ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8123
)

### HTTPS connection (without ssl peer verification) ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8443, https=TRUE, ssl_verifypeer=FALSE
)

## Write a table in the database ----

library(dplyr)
data("mtcars")
mtcars <- as_tibble(mtcars, rownames="car")
dbWriteTable(con, "mtcars", mtcars)

## Query the database ----

carsFromDB <- dbReadTable(con, "mtcars")
dbGetQuery(con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110")

## By default, ClickHouseHTTP relies on the
## Apache Arrow format provided by ClickHouse.
## The `format` argument of the `dbGetQuery()` function can be used to
## rely on the *TabSeparatedWithNamesAndTypes* format.
selCars <- dbGetQuery(
  con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110",
  format="TabSeparatedWithNamesAndTypes"
)
## Identifying the original ClickHouse data types
attr(selCars, "type")
```

```

## Using alternative databases stored in ClickHouse ----

dbSendQuery(con, "CREATE DATABASE swiss")
dbSendQuery(con, "USE swiss")

## The chosen database is used until the session expires.
## It can also be chosen when connecting using the `dbname` argument of
## the `dbConnect()` function.

## The example below shows that spaces in column names are supported.
## It also shows the support of R `list` using the *Array* ClickHouse type.
data("swiss")
swiss <- as_tibble(swiss, rownames="province")
swiss <- mutate(swiss, "pr letters"=strsplit(province, ""))
dbWriteTable(
  con, "swiss", swiss,
  engine="MergeTree() ORDER BY (Fertility, province)"
)
swissFromDB <- dbReadTable(con, "swiss")

## A table from another database can also be accessed as following:
dbReadTable(con, SQL("default.mtcars"))

## End(Not run)
## Not run:

## Connection ----

library(DBI)
### HTTP connection ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8123
)

### HTTPS connection (without ssl peer verification) ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8443, https=TRUE, ssl_verifypeer=FALSE
)

## Write a table in the database ----

library(dplyr)
data("mtcars")
mtcars <- as_tibble(mtcars, rownames="car")
dbWriteTable(con, "mtcars", mtcars)

## Query the database ----

```

```
carsFromDB <- dbReadTable(con, "mtcars")
dbGetQuery(con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110")

## By default, ClickHouseHTTP relies on the
## Apache Arrow format provided by ClickHouse.
## The `format` argument of the `dbGetQuery()` function can be used to
## rely on the *TabSeparatedWithNamesAndTypes* format.
selCars <- dbGetQuery(
  con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110",
  format="TabSeparatedWithNamesAndTypes"
)
## Identifying the original ClickHouse data types
attr(selCars, "type")

## Using alternative databases stored in ClickHouse ----

dbSendQuery(con, "CREATE DATABASE swiss")
dbSendQuery(con, "USE swiss")

## The chosen database is used until the session expires.
## It can also be chosen when connecting using the `dbname` argument of
## the `dbConnect()` function.

## The example below shows that spaces in column names are supported.
## It also shows the support of R `list` using the *Array* ClickHouse type.
data("swiss")
swiss <- as_tibble(swiss, rownames="province")
swiss <- mutate(swiss, "pr letters"=strsplit(province, ""))
dbWriteTable(
  con, "swiss", swiss,
  engine="MergeTree() ORDER BY (Fertility, province)"
)
swissFromDB <- dbReadTable(con, "swiss")

## A table from another database can also be accessed as following:
dbReadTable(con, SQL("default.mtcars"))

## End(Not run)
## Not run:

## Connection ----

library(DBI)
### HTTP connection ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8123
)

### HTTPS connection (without ssl peer verification) ----
```

```

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8443, https=TRUE, ssl_verifypeer=FALSE
)

## Write a table in the database ----

library(dplyr)
data("mtcars")
mtcars <- as_tibble(mtcars, rownames="car")
dbWriteTable(con, "mtcars", mtcars)

## Query the database ----

carsFromDB <- dbReadTable(con, "mtcars")
dbGetQuery(con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110")

## By default, ClickHouseHTTP relies on the
## Apache Arrow format provided by ClickHouse.
## The `format` argument of the `dbGetQuery()` function can be used to
## rely on the *TabSeparatedWithNamesAndTypes* format.
selCars <- dbGetQuery(
  con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110",
  format="TabSeparatedWithNamesAndTypes"
)
## Identifying the original ClickHouse data types
attr(selCars, "type")

## Using alternative databases stored in ClickHouse ----

dbSendQuery(con, "CREATE DATABASE swiss")
dbSendQuery(con, "USE swiss")

## The chosen database is used until the session expires.
## It can also be chosen when connecting using the `dbname` argument of
## the `dbConnect()` function.

## The example below shows that spaces in column names are supported.
## It also shows the support of R `list` using the *Array* ClickHouse type.
data("swiss")
swiss <- as_tibble(swiss, rownames="province")
swiss <- mutate(swiss, "pr letters"=strsplit(province, ""))
dbWriteTable(
  con, "swiss", swiss,
  engine="MergeTree() ORDER BY (Fertility, province)"
)
swissFromDB <- dbReadTable(con, "swiss")

## A table from another database can also be accessed as following:
dbReadTable(con, SQL("default.mtcars"))

```



```
## End(Not run)
```

```
ClickHouseHTTPDriver-class
```

```
Driver for the ClickHouse database using HTTP(S) interface
```

Description

Driver for the ClickHouse database using HTTP(S) interface

Connect to a ClickHouse database using the ClickHouseHTTP DBI

Usage

```
## S4 method for signature 'ClickHouseHTTPDriver'
dbConnect(
  drv,
  host = "localhost",
  port = 8123L,
  dbname = "default",
  user = "default",
  password = "",
  https = FALSE,
  ssl_verifypeer = TRUE,
  session_timeout = 3600L,
  convert_uint = TRUE,
  ...
)
```

Arguments

<code>drv</code>	A driver object created by <code>ClickHouseHTTP()</code>
<code>host</code>	name of the database host (default: "localhost")
<code>port</code>	port on which the database is listening (default: 8123L)
<code>dbname</code>	name of the default database (default: "default")
<code>user</code>	user name (default: "default")
<code>password</code>	user password (default: "")
<code>https</code>	a logical to use the HTTPS protocol (default: FALSE)
<code>ssl_verifypeer</code>	a logical to verify SSL certificate when using HTTPS (default: TRUE)
<code>session_timeout</code>	timeout in seconds (default: 3600L seconds)
<code>convert_uint</code>	a logical: if TRUE (default), UInt ClickHouse data types are converted in the following R classes: <ul style="list-style-type: none"> • <code>UInt8</code> : logical

- UInt16: Date (when using Arrow format in `dbSendQuery`, `ClickHouseHTTPConnection`, `character-method`)
- UInt32: POSIXct (when using Arrow format in `dbSendQuery`, `ClickHouseHTTPConnection`, `character-method`)

... Other parameters passed on to methods

Value

A ClickHouseHTTPConnection

See Also

[ClickHouseHTTPConnection](#)

Examples

```
## Not run:

## Connection ----

library(DBI)
### HTTP connection ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8123
)

### HTTPS connection (without ssl peer verification) ----

con <- dbConnect(
  ClickHouseHTTP::ClickHouseHTTP(), host="localhost",
  port=8443, https=TRUE, ssl_verifypeer=FALSE
)

## Write a table in the database ----

library(dplyr)
data("mtcars")
mtcars <- as_tibble(mtcars, rownames="car")
dbWriteTable(con, "mtcars", mtcars)

## Query the database ----

carsFromDB <- dbReadTable(con, "mtcars")
dbGetQuery(con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110")

## By default, ClickHouseHTTP relies on the
## Apache Arrow format provided by ClickHouse.
## The `format` argument of the `dbGetQuery()` function can be used to
## rely on the *TabSeparatedWithNamesAndTypes* format.
selCars <- dbGetQuery(
```

```

    con, "SELECT car, mpg, cyl, hp FROM mtcars WHERE hp>=110",
    format="TabSeparatedWithNamesAndTypes"
  )
  ## Identifying the original ClickHouse data types
  attr(selCars, "type")

  ## Using alternative databases stored in ClickHouse ----

  dbSendQuery(con, "CREATE DATABASE swiss")
  dbSendQuery(con, "USE swiss")

  ## The chosen database is used until the session expires.
  ## It can also be chosen when connecting using the `dbname` argument of
  ## the `dbConnect()` function.

  ## The example below shows that spaces in column names are supported.
  ## It also shows the support of R `list` using the *Array* ClickHouse type.
  data("swiss")
  swiss <- as_tibble(swiss, rownames="province")
  swiss <- mutate(swiss, "pr letters"=strsplit(province, ""))
  dbWriteTable(
    con, "swiss", swiss,
    engine="MergeTree() ORDER BY (Fertility, province)"
  )
  swissFromDB <- dbReadTable(con, "swiss")

  ## A table from another database can also be accessed as following:
  dbReadTable(con, SQL("default.mtcars"))

  ## End(Not run)

```

ClickHouseHTTPResult-class

ClickHouseHTTPResult class.

Description

ClickHouseHTTPResult class.

Index

ClickHouseHTTP, [2](#)
ClickHouseHTTP(), [9](#)
ClickHouseHTTPConnection, [10](#)
ClickHouseHTTPConnection-class, [2](#)
ClickHouseHTTPODriver, [2](#)
ClickHouseHTTPODriver-class, [9](#)
ClickHouseHTTPResult, [5](#)
ClickHouseHTTPResult-class, [11](#)

dbConnect(), [3](#)
dbConnect, ClickHouseHTTPODriver-method
(ClickHouseHTTPODriver-class), [9](#)
dbCreateTable, ClickHouseHTTPConnection-method
(ClickHouseHTTPConnection-class),
[2](#)
dbGetInfo, ClickHouseHTTPConnection-method
(ClickHouseHTTPConnection-class),
[2](#)
dbSendQuery, ClickHouseHTTPConnection, character-method,
[10](#)
dbSendQuery, ClickHouseHTTPConnection, character-method
(ClickHouseHTTPConnection-class),
[2](#)
dbWriteTable, ClickHouseHTTPConnection, ANY-method
(ClickHouseHTTPConnection-class),
[2](#)